© **ISO 1999 – All rights reserved**

ISO/TC 184/SC 4

Date: 1999-9-9

ISO 10303-11:1994/DAM 1

ISO/TC 184/SC 4/WG 11 N084

THIS DOCUMENT IS WG11 N084 AND IS A WORKING DRAFT, IT HAS THE FORM OF THE DAM BUT IS NOT THE ACTUAL DAM.

Industrial automation systems and integration — Product data representation and exchange — Part 11: Implementation methods: The EXPRESS language reference manual

AMENDMENT 1

Type of document: International Standard
Sub-type of document: Amendment
Stage of document:
Language of document: E

Copyright notice

This ISO document is a draft amendment and is copyright-protected by ISO. Except as permitted under the applicable laws of the user's country, neither this ISO draft nor any extract from it may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, photocopying, recording or otherwise, without prior written permission being secured.

Requests for permission to reproduce should be addressed to ISO at the address below or ISO's member body in the country of the requester:

Copyright Manager
ISO Central Secretariat
1 rue de Varembé
1211 Geneva 20 Switzerland
tel. + 41 22 749 0111
fax + 41 22 734 0179
e-mail central@iso.ch

Reproduction may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Amendment 1 to International Standard ISO 10303-11:1994 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

Introduction

This document amends ISO 10303-11:1994/Cor.2:1999; Industrial automation systems and integration — Product data representation and exchange — Part 11: Implementation methods: The EXPRESS language reference manual. The amended document supercedes ISO 10303-11:1994/Cor.2:1999.

The modifications to the text of ISO 10303-11:1994/Cor.2:1999 are of one kind. Each modification is identified as to which kind it represents and is marked with a diamond (♦). The one kind of modification is:

♦ (CHANGE): A change to the requirements of ISO 10303-11:1994. Such a modification affects the conformance of implementations.

All changes and additions to all clauses facilitate the development and implementation of the ISO 10303-1000 series of parts called application modules. The scope of this amendment was driven by requirements from the development of the application modules architecture and from the development of application modules themselves. It is expected that the changes and additions will also be beneficial to all users of ISO 10303-11.

Every attempt has been made to keep this amendment from invalidating any EXPRESS schema that conforms to ISO 10303-11:1994/Cor.2:1999. This was a primary requirement on the amendment that limited its scope and technical content. However, several new words have been added to the language: `based_on`, `end_subtype_constraint`, `extensible`, `generic_entity`, `renamed`, `subtype_constraint`, `total_over` and `with`. Schemas containing these words as EXPRESS identifiers become invalid under this amendment.

> NOTE - This amendment is limited in scope and does not eliminate the requirement for the next edition of ISO 10303-11. The technical content of this amendment was written to be compatible with and places constraints on the next edition of ISO 10303-11.

Clause 7.2.1

♦ (CHANGE) Add the new keywords EXTENSIBLE, BASED_ON, WITH, GENERIC_ENTITY, SUBTYPE_CONSTRAINT, TOTAL_OVER and RENAMED to table 1.

Clause 7.6

♦ (CHANGE) Add a new clause 7.6 after clause 7.5 as follows:

7.6 Lanuage version identification

The language version identifier specifies the the edition of the EXPRESS Language Reference Manual upon which a schema is based. This amendment to EXPRESS is version 4.

NOTE 1 - Although the language version identifier is optional under this amendment, its use is strongly encouraged when using any capabilities added to EXPRESS by this amendment.

EXAMPLE 14a - The following EXPRESS uses the language version identifier:

```
{iso standard 10303 part (11) version (4)}
SCHEMA configuration_controlled_design_of_the_future;
END_SCHEMA;
```

Syntax:

```
405 language_version_id = '{ iso standard 10303 part (11) version (4) }' .
302 syntax = [ language_version_id ] schema_decl { schema_decl } .
```

Clause 8.4

♦ (CHANGE) Replace the existing 8.4 with the following:

8.4 Constructed data types

There are two kinds of constructed data types in EXPRESS: `ENUMERATION` data types and `SELECT` data types. These data types have similar syntactic structures and may only be used to provide underlying representations of defined data types.

8.4.1 Enumeration data type

An `ENUMERATION` data type has as its domain a set of names. The names are the only valid values of the enumeration data type. An **enumeration_id** is used to designate these names that are referred to as enumeration items. An `ENUMERATION` data type may be extensible or not.

An `ENUMERATION` data type that is not extensible and that is not based on an extensible `ENUMBERATION` data type has as its domain an ordered set of enumeration items in its declaration.

An extensible `ENUMERATION` data type has as its domain the set of the enumeration items in its declaration plus the union of the sets of enumeration items comprising the domains of all enumeration data types based on the extensible enumeration data type. An enumeration data type that is based on an extensible enumeration data type has as its domain the set of enumeration items in its declaration plus the enumeration items specified directly, not via extension, in the extensible enumeration data type upon which it is based. An enumeration data type may be both an extensible enumeration data type and be based on another extensible enumeration data type. An extensible enumeration data type is a generalisation of the enumeration data types that are based on it. An extensible `ENUMERATION` data type is

specified using the EXTENSIBLE reserved word and an ENUMERATION data type based on an extensible ENUMERATION data type is specified using the BASED_ON reserved word. An extensible ENUMERATION may be specified without enumeration items and may be based on another extensible ENUMERATION and not specify any enumeration items extending that base ENUMERATION.

Two enumeration data types may extend the same extensible enumeration data types with the same enumeration_id. In this case, the enumeration_is within the domain of the extensible enumeration and both extensions and designates the same value.

NOTE 1 - This means that an extensible enumeration which is extended two or more times in a single context may have a larger domain than its extensions, in which case it really is a generalisation.

EXAMPLE 34 - The following EXPRESS results in a single enumeration item named "red" as both stop_light and canadian_flag extend the domain of colour.

```
TYPE colour = EXTENSIBLE ENUMERATION;
END_TYPE;

TYPE stop_light = ENUMERATION OF (red, yellow, green) BASED_ON colour;
END_TYPE;

TYPE canadian_flag = ENUMERATION OF (red, white) BASED_ON color;
END_TYPE;
```

Syntax:

```
201 enumeration_type = [ EXTENSIBLE ] ENUMERATION [ (( OF enumeration_items ) |
enumeration_extension) ] .
402 enumeration_items = '(' enumeration_id { ',' enumeration_id } ')' .
403 enumeration_extension = BASED_ON type_ref [ WITH enumeration_items ] .
```

Two different ENUMERATION data types may declare the same **enumeration_id** defined in different domains. In this case, any reference to the **enumeration_id** (e.g. in an expression) shall be pre-qualified with the data type identifier to ensure that the reference is unambiguous. The reference then appears as: **type_id.enumeration_id**.

NOTE 2 - There is always a **type_id** available, because an ENUMERATION data type is only allowed as the underlying representation of a defined data type.

NOTE 3 - In the previous edition of this standard the ordering of the enumeration items implied some value ordering this is not the case in this edition of the standard, except as noted in rule (d) below. This has been changed to allow for extensible enumeration data types, in which the ordering of the extensions cannot be determined.

Rules and Restrictions

a. An enumeration data type shall only be used as the underlying data type of a defined data type.
b. Only defined data types whose underlying data types are extensible enumerations may be extended.
c. An enumeration extension may itself be an extensible specification if the reserved word EXTENSIBLE is specified in the definition of the enumeration.
d. For comparison purposes, the ordering of the values of an enumeration that is not extensible and that is not based on an extensible enumeration may be determined by their relative position in the **enumeration_id** list; the first occurring item shall be less than the second; the second less than the

third, etc.

e. There is no ordering of the values of an extensible enumeration or an enumeration that is based on an extensible enumeration.
f. An enumeration that is not extensible and not based on an extensible enumeration shall specify enumeration items as its domain.
g. An enumeration that is not extensible and that is based on an extensible enumeration shall specify enumeration items that extend the domain of the extensible enumeration upon which it is based.
h. The defined data type which declares the enumeration data type shall not include a domain (where) rule.

NOTE 4 - The above rules ensure that a defined data type names an enumeration data type, and the defined data type is not a specialisation of the enumeration data type.

EXAMPLE 34a - This example uses `ENUMERATION` data types to show how different kinds of vehicles might travel.

```
TYPE car_can_move = ENUMERATION OF (left, right, backward, forward);
END_TYPE;

TYPE plane_can_move = ENUMERATION OF (left, right, backward, forward, up, down);
END_TYPE;
```

The enumeration item **left** has two independent definitions, one being given by each data type of which it is a component. There is no connection between these two definitions of the identifier **left**. A reference to **left** or **right**, by itself, is ambiguous. To resolve the ambiguity, a reference to either of these values must be qualified by the data type name, e.g., **car_can_move.left**.

EXAMPLE 34b - The following example shows how an extensible enumeration may be used to model a context dependent concept of approval. **general_approval**, as its name suggests, is the most general concept of approval explicitly defining only two values. By declaring **general_approval** to be an extensible enumeration allows it to take on context dependent values in schemas which declare extensions to it. If used to represent the domain of attribute then the allowed values of the attribute are context dependent.

```
SCHEMA s1;

TYPE general_approval = EXTENSIBLE ENUMERATION OF (approved, rejected);
END_TYPE;

END_SCHEMA;

SCHEMA s2;

USE FROM s1 (general_approval);

TYPE domain2_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (pending);
END_TYPE;


END_SCHEMA;


SCHEMA s3;


USE FROM S1 (general_approval);

TYPE domain3_approval = EXTENSIBLE ENUMERATION BASED_ON general_approval WITH (cancelled);
END_TYPE;
```

```
END_SCHEMA;

SCHEMA s4;

USE FROM s2 (domain2_approval);

REFERENCE FROM S3 (domain3_approval);

TYPE specific_approval = ENUMERATION BASED_ON domain2_approval WITH (rework);
END_TYPE;

END_SCHEMA;
```

In the context of schema **S1:**

- **general_approval** has the domain (**approved**, **rejected**).

In the context of schema **S2**:

- **general_approval** has the domain (**approved**, **rejected**, **pending**);
- **domain2_approval** has the domain (**approved**, **rejected**, **pending**).

In the context of schema **S3**:

- **general_approval** has the domain (**approved**, **rejected**, **cancelled**);
- **domain3_approval** has the domain (**approved**, **rejected**, **cancelled**).

In the context of schema **S4**:

- **general_approval** has the domain (**approved**, **rejected**, **pending**, **cancelled**, **rework**);
- **domain2_approval** has the domain (**approved**, **rejected**, **pending**, **rework**);
- **domain3_approval** has the domain (**approved**, **rejected**, **cancelled**);
- **specific_approval** has the domain (**approved**, **rejected**, **pending**, **rework**).

8.4.2 Select data type

A SELECT data type is a generalization of the named data types in its domain. All named data types in the domain of a SELECT data type are compatible with the SELECT data type. A SELECT data type may be extensible or not.

A SELECT data type that is not extensible and that is not based on an extensible SELECT data type has as its domain the union of the domains of the named data types in its select list.

An extensible SELECT data type has as its domain the union of the domains of the named data types in its select list plus the union of the domains of all select data types based on the extensible select data type. A select data type that is based on an extensible select data type has as its domain the union of the named data types in its select list plus the union of the named data types specified directly, not via extension, in the extensible select data type upon which it is based. A select data type may be both an extensible select data type and be based on another extensible select data type. An extensible SELECT data type is specified using the EXTENSIBLE reserved word and a SELECT data type based on an extensible SELECT data type is

specified using the BASED_ON reserved word. An extensible SELECT may be specified without a select list and may be based on another extensible SELECT and not specify a select list extending that base SELECT.

A SELECT data type may be constrained to have only entity instances in its domain by using the reserved word GENERIC_ENTITY. In this case all select elements must either be entity data types or select data types whose select list only includes entity data types. If an extensible select data type is constrained to be a generic-entity select data type, then all extensions of that select data type shall be generic-entity select data types, and shall specify the GENERIC_ENTITY reserved word.

Syntax:

```
284 select_type = [ EXTENSIBLE ] [ GENERIC_ENTITY ] SELECT [( select_list |
select_extension ) ] .
410 select_list = '(' named_types { ',' named_types } ')' .
409 select_extension = BASED_ON type_ref [ WITH select_list ] .
```

Rules and Restrictions

a. Each item in the select list shall be an entity data type or a defined data type.
b. A SELECT data type shall only be used as the underlying data type of a defined data type.
c. Only defined data types who's underlying data types are extensible selects may be extended.
d. A select extension may itself be an extensible specification if the reserved word EXTENSIBLE is specified in the definition of the select.
e. If the reserved word GENERIC_ENTITY is used then only generic-entity elements are valid in the select list, where generic-entity element is defined as being either an entity data type or a select of generic-entity elements.
f. A select that is not extensible and not based on an extensible select shall specify a select list as its domain.
g. A select that is not extensible and that is based on an extensible select shall specify a select list that extend the domain of the extensible select upon which it is based.

NOTE - The value of a SELECT data type may be a value of more than one of the named data types specified in the select list for that select data type.

EXAMPLE 35 - If **a** and **b** are subtypes of **c**, and if they are related by an ANDOR expression, and if we have a data type defined by **SELECT (a,b)**, then we may have the value of the SELECT data type which is an **a** and a **b** at the same time.

EXAMPLE 36 - A choice must be made among several types of things in a given context.

```
TYPE attachment_method = EXTENSIBLE SELECT(nail, screw);
END_TYPE;

TYPE permanent_attachment = SELECT BASED ON attachment_method WITH (glue, weld);
END_TYPE;

ENTITY nail;
length : REAL;
head_area : REAL;
END_ENTITY;

ENTITY screw;
length : REAL;
pitch : REAL;
END_ENTITY;

ENTITY glue;
composition : material_composition;
solvent : material_composition;
END_ENTITY;

ENTITY weld;
composition : material_composition;
END_ENTITY;

ENTITY wall_mounting;
mounting : product;
on : wall;
using : attachment_method;
END_ENTITY;
```

A **wall_mounting** attaches a **product** onto a **wall** using an attachment method. The initial attachment method describes temporary attachment methods. These methods are then extended to add permament attachement methods. A value of a **wall_mounting** will have an **using** attribute which is a value of one of **nail**, **screw**, **glue** or **weld**.

Clause 8.5

♦ (CHANGE) Add `generic_entity_type` to the end of the list of data types in rule 211 and add the following as the next to last sentence of paragraph 1:

A `GENERIC_ENTITY` data type is a generalisation of all entity data types and a subtype of the `GENERIC` data type.

Clause 8.6

♦ (CHANGE) Add a dot in table 7 under the column labeled "a" in the row labeled "Generalized Data Types"

Clause 8.6.1

♦ (CHANGE) Replace the second sentence with:

The base data types are the concrete data types and the generalized data types. The concrete types are potentially instantiable when used to define the domain of an attribute, the generalized data types are not

instantiable when used to define the domain of an attribute. The concrete data types are the simple data types, the aggregation data types and the named data types.

♦ (CHANGE) Add a third paragraph:

The generalized data types shall only be used as base data types in an abstract entity (see 9.7.1.1).

♦ (CHANGE) Replace rule 171 with the following rule:

```
171 base_type = concrete_types | generalized_types .
```

♦ (CHANGE) Add the following rule after rule 171:

```
401 concrete_types = aggregation_types | simple_types | named_types .
```

Clause 9

♦ (CHANGE) Add "Subtype_constraint;" after "Entity;" to the list of principle capabilities in the second paragraph.

Clause 9.2

♦ (CHANGE) Replace rule 197 with:

```
197 entity_head = ENTITY entity_id [ABSTRACT] [subsuper] ';' .
```

Clause 9.2.1

♦ (CHANGE) Add the following after the second sentence in the first paragraph as part of the first paragraph:

In the case of an abstract entity whose explicit or derived attributes have `GENERIC`, `GENERIC_ENTITY` or `AGGREGATE` as their data type, the attributes must be redeclared with instantiable data types in at least one subtype of the abstract entity in order to create an instance of the abstract entity data type.

Clause 9.2.1.1

♦ (CHANGE) Replace rule 167 with:

```
167 attribute_decl = attribute_id | redeclared_attribute .
```

Clause 9.2.1.2

♦ (CHANGE) Replace rule 167 with:

```
167 attribute_decl = attribute_id | redeclared_attribute .
```

Clause 9.2.3.4

♦ (CHANGE) Change "five" (ISO 10303-11:Cor.2:1999 changed "three" to "five" and added a new second and third list item) to "following" in the third sentence of the first paragraph and add the following as the last two items in the list after the first paragraph:

- if the original data type of the attributes is a generic-entity data type, then the data type may be changed to a generic-entity select data type;
- an attribute in the supertype may be given a new identifier in the subtype. The new attribute identifier obeys all the scope and visibility rules defined in clause 10.

♦ (CHANGE) Add the following note after the list after the first paragraph:

NOTE - The declaration of a new identifier does not remove the old identifier from the name scope. The old identifier remains available in this entity and any subtypes declared for this entity.

♦ (CHANGE) Add the following to the Syntax before rule 262.

```
406 redeclared_attribute = qualified_attribute [RENAMED attribute_id ] .
```

♦ (CHANGE) Add the following after the last item in the list under Rules and restrictions (ISO 10303-11:Cor.2:1999 added a new (c) and renumbered the previous (c) to (d)):

e) If the attribute is given a new identifier then that identifier shall not be the same as the identifier for any attribute in any supertype of the current entity.

♦ (CHANGE) Replace example 49 with the following:

EXAMPLE 49 - Some geometry systems use floating point coordinates while others work in an integer coordinate space.

```
ENTITY point;
  x : NUMBER;
  y : NUMBER;
END_ENTITY;

ENTITY integer_point
  SUBTYPE OF (point);
  SELF\point.x RENAMED integer_x : INTEGER;
  SELF\point.y RENAMED integer_y : INTEGER;
END_ENTITY;
```

Clause 9.2.4

♦ (CHANGE) Add the following informative note before the first paragraph:

NOTE - In order that existing schemas remain valid, the declaration of subtype/supertype constraints within the declaration of an entity remains valid under this amendment to ISO 10303-11. However, its use is deprecated and its removal is planned in future editions of ISO 10303-11. The use of the `SUBTYPE_CONSTRAINT` (see 9.7) in new schema development is encouraged.

Clause 9.2.6

♦ (CHANGE) Add the following list item before the last list item in the list after the first paragraph:

- a constructed data type that is based on an extensible constructed data type is a specialization of that extensible constructed data type.

Clause 9.3

♦ (CHANGE) Add the following sentences to the end of the first paragraph:

A schema may undergo changes within a development or standardisation environment. To support the capability of referring to a particular version of a schema, a schema version identifier concept is provided. This standard does not prescribe the format of the schema version identifier except to define it to be a string literal.

♦ (CHANGE) Add the following example after the first paragraph:

EXAMPLE - There may be multiple versions of schemas and the language version identifier may be included as well.

```
SCHEMA geometry_schema version_1;
END_SCHEMA;

SCHEMA geometry_schema version_2;
END_SCHEMA;

SCHEMA support_resource_schema {ISO standard 10303 part(41) object(1) version(8)};
END_SCHEMA;

{ISO standard 10303 part(11) version(4)} SCHEMA support_resource_schema {ISO standard 10303
part(41) object(1) version(9)};
END_SCHEMA;
```

♦ (CHANGE) Replace rule 281 with the following:

```
281 schema_decl = SCHEMA schema_id [ schema_version_id ] ';' schema_body END_SCHEMA
';' .
```

♦ (CHANGE) Add the rule following to the Syntax after rule 281:

```
407 schema_version_id = string_literal .
```

♦ (CHANGE) Add "`subtype_constraint_decl`" as an option in rule 189 after "`entity_decl`"

Clause 9.7

♦ (CHANGE) After Clause 9.6 add the following as Clause 9.7:

9.7 Subtype constraints

The concepts of subtypes and supertypes are specified in 9.2.3 and 9.2.4 specifies the concept of subtype/supertype constraints. It is possible to specify constraints on which subtype/supertype graphs may be instantiated outside the declaration of a entity. This may be specified by declaring a

SUBTYPE_CONSTRAINT.

Syntax:

```
412 subtype_constraint_decl = subtype_constraint_head subtype_constraint_body
END_SUBTYPE_CONSTRAINT ';' .
413 subtype_constraint_head = SUBTYPE_CONSTRAINT subtype_constraint_id FOR
entity_ref ';' .
411 subtype_constraint_body = [ abstract_supertype ] [ total_over ] [
supertype_expression ';' ] .
400 abstract_supertype = ABSTRACT SUPERTYPE ';' .
415 total_over = TOTAL_OVER '(' entity_ref { ',' entity_ref } ')' ';' .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .
301 supertype_term = entity_ref | one_of | '(' supertype_expression ')' .
250 one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .
```

The SUBTYPE_CONSTRAINT is used to specify the following constraints on the possible supertype/subtype instantiation:

- if the supertype is abstract and should only be instantiated through its subtypes;
- if a collection of the subtypes for the supertype provides a total coverage, i.e., if a total coverage is specified then an instance of any subtype of the supertype shall also be an instance of at least one of the subtypes specified in the total over specification.
- the relationship between some of the subtypes.

Each of these different areas will be discussed in more detail in the following subclauses. Annex B provides the formal approach to determining the potential combinations of subtype/supertype which may be instantiated under the several possible constraints that are described below.

9.7.1 Abstract entities and constraints

EXPRESS allows for the declaration of entity data types that are not intended to be directly instantiated and may only be instantiated through their subtypes. There are two kinds, abstract entity data type and abstract supertype constraint.

9.7.1.1 Abstract entity data type

An abstract entity data type may declare explicit and derived attributes of data type generalized_types, these may then be redeclared as being of an instantiable data type in subtypes of the abstract entity data type. If a subtype of an abstract entity data type is itself an abstract entity data type it need not redeclare the uninstantiable inherited attributes to be of instantiable data types. In a subtype of an abstract entity data type, that is not itself an abstract entity data type, no inherited or directly declared attributes may be of an uninstantiable data type.

Rules and Restrictions

a. An abstract entity data type declaration includes the ABSTRACT keyword in the entity data type declaration.
b. An abstract entity data type is not instantiable unless it is part of a complex entity data type where all

attributes of data type `generalized_types` have been redeclared to be of an instantiable data type.

EXAMPLE 70a - In a general approval model we may wish to identify that a collection of things may be approved. This model could then be used in a number of other schemas and refined to identify the actual items approved.

```
ENTITY general_approval ABSTRACT;
approved_items : BAG OF GENERIC_ENTITY;
status : approval_status;
END_ENTITY;
```

9.7.1.2 Abstract supertype constraint

The abstract supertype declaration specified in 9.2.4.1 may also be declared in a `SUBTYPE_CONSTRAINT`.

Rules and Restrictions

a. An abstract supertype is defined by a `SUBTYPE_CONSTRAINT` for the supertype including the `ABSTRACT SUPERTYPE` keywords.

EXAMPLE 70b - In a general classification model we may wish to identify an entity called class, which in this context is instantiable. In a more specific model we may wish to use the class entity, but constrain it such that it may only be instantiated through its locally declared subtypes.

```
SCHEMA general_classification_model;

ENTITY class;
name : class_name;
END_ENTITY;


END_SCHEMA;

SCHEMA specific_classification_model;
USE FROM general_classification_model;

ENTITY class_of_facility
SUBTYPE OF (class);
END_ENTIY;

ENTITY class_of_organisation
SUBTYPE OF (class);
END_ENTITY;

SUBTYPE_CONSTRAINT independent_classification FOR class;
ABSTRACT SUPERTYPE;
ONEOF(class_of_facility, class_of_organisation);
END_SUBTYPE_CONSTRAINT;

END_SCHEMA;
```

9.7.2 Total coverage subtypes

A supertype entity may be declared to be fully defined for one context by a collection of its subtypes.

EXAMPLE 70c - The concept person is totally covered by the concepts of male and female, there may be other concepts but each person is either male or female. Therefore we can say that person is totally covered by male and female.

In the case of two or more subtype constraints specifying TOTAL_OVER constraints for the same entity data type then these TOTAL_OVER constraints are considered in combination, i.e., TOTAL_OVER (a,b) and TOTAL_OVER (c,d) must both be satisfied.

Rules and Restrictions

a. All subtypes specified in the TOTAL_OVER constraint for a given supertype Sup shall be direct subtypes of Sup.
b. Other subtypes, however defined or constrained, must always be combined with at least one of the subtypes in the TOTAL_OVER specification;
c. Since a supertype may have more than one context, it may also be constrained to have more than one TOTAL_OVER constraint.

EXAMPLE 70d - The following specifies that each person must be either a male or a female. The example says nothing about the relationship between male and female, and we could create an instance which is both male and female. The subtype employee must always be combined with the concepts of male and female and can never be instantiated independently.

```
ENTITY person;
name : personal_name;
END_ENTITY;

ENTITY male
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY female
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY employee
SUBTYPE OF (person);
...
END_ENTITY;

SUBTYPE_CONSTRAINT person_sex FOR person;
ABSTRACT SUPERTYPE;
TOTAL_OVER (male, female);
END_SUBTYPE_CONSTRAINT;
```

9.7.3 Overlapping subtypes and their specification

Two or more direct subtypes of a particular supertype may be allowed to have overlapping instantiations for a particular context. The SUBTYPE_CONSTRAINT specification may be used to specify how a particular group of direct subtypes are related.

9.7.3.1 Oneof

The ONEOF constraint specified in 9.2.4.2 may be declared in a SUBTYPE_CONSTRAINT. The ONEOF

constraint may be combined with the other supertype constraints to enable the writing of complex constraints.

NOTE - The phrase `ONEOF(a,b,c)` reads in natural language as "an instance shall consist of one and only one of the entity data types a,b,c".

EXAMPLE 70e - An instance of a supertype may be established through the instantiation of only one of its subtypes. This constraint is declared using the `ABSTRACT` and `ONEOF` constraints. There are many kinds of pet, but no single pet can be simultaneously two or more kinds of pet.

```
ENTITY pet
name : pet_name;
...
END_ENTITY;

SUBTYPE_CONSTRAINT separate_species FOR pet;
ABSTRACT SUPERTYPE;
ONEOF(cat, rabbit, dog, ... );
END_SUBTYPE_CONSTRAINT;

ENTITY cat
SUBTYPE OF (pet);
...
END_ENTITY;

ENTITY rabbit
SUBTYPE OF (pet);
...
END_ENTITY;

ENTITY dog
SUBTYPE OF (pet);
...
END_ENTITY;
```

9.7.3.2 Andor

The `ANDOR` constraint specified in 9.2.4.3 may be declared in a `SUBTYPE_CONSTRAINT`.

EXAMPLE 70f - A person could be an employee who is taking night classes and may therefore be simultaneously both an employee and a student.

```
ENTITY person
...
END_ENTITY;

SUBTYPE_CONSTRAINT employee_maybe_student FOR person;
employee ANDOR student;
END_SUBTYPE_CONSTRAINT;

ENTITY employee
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY student
SUBTYPE OF (person);
...
END_ENTITY;
```

9.7.3.3 And

The AND constraint specified in 9.2.4.4 may be declared in a SUBTYPE_CONSTRAINT.

EXAMPLE 70g - A person could be categorized into being either male or female, and could also be categorized into being either a citizen or an alien.

```
ENTITY person;
...
END_ENTITY;

SUBTYPE_CONSTRAINT no_mixing FOR person;
SUPERTYPE OF (ONEOF(male,female) AND
ONEOF(citizen,alien));
END_SUBTYPE_CONSTRAINT;

ENTITY male
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY female
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY citizen
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY alien
SUBTYPE OF (person);
...
END_ENTITY;
```

Clause 10

♦ (CHANGE) Add a new row to table 9 after the row with the value "schema" in the column labeled "Item". The value for the column labeled "Item" is "subtype constraint", the value for the column labeled "Scope" is a dot and the value for the column labeled "Identifier" is a dot.

Clause 10.3.13

♦ (CHANGE) Add the following to the list under "Declarations" after "rule":

- subtype constraint;

Clause 10.3.17

♦ (CHANGE) Add a new clause 10.3.17 after clause 10.3.16 as follows:

10.3.17 Subtype constraint

Visibility : A subtype constraint identifier is visible in the scope of the schema in which it is declared.

NOTE - The subtype constraint identifier is of use only to an implementation. EXPRESS provides no mechanism for referencing subtype constraint identifiers.

Scope : A subtype constraint extends the scope of the entity for which it is declared. This scope extension extends from the keyword `SUBTYPE_CONSTRAINT` to the keyword `END_SUBTYPE_CONSTRAINT` which terminates that subtype constraint declaration.

Clause 11.1

♦ (CHANGE) In the second sentence of the first paragraph replace "The `USE` specification gives the name" with "The `USE` specification gives the name, and optionally the version,"

♦ (CHANGE) Replace rule 313 with the following:

```
313 use_clause = USE FROM schema_ref [ schema_version_ref ] [ '('
named_type_or_rename { ',' named_type_or_rename } ')' ] ';' .
```

♦ (CHANGE) Add the following rule after rule 313:

```
408 schema_version_ref = string_literal .
```

Clause 11.2

♦ (CHANGE) In the second sentence of the second paragraph replace "The `REFERENCE` specification gives the name" with "The `REFERENCE` specification gives the name, and optionally the version,"

♦ (CHANGE) Replace rule 313 with the following:

```
267 reference_clause = REFERENCE FROM schema_ref [ schema_version_ref ] [ '('
resource_or_rename { ',' resource_or_rename } ')' ] ';' .
```

♦ (CHANGE) Add the following rule after rule 267:

```
408 schema_version_ref = string_literal .
```

Clause 11.4.3

♦ (CHANGE) Add the following to the list in the first sentence of the first paragraph after the list item beginning "- all rules":

- all subtype constraints for the interfaced entity data type;

Clause 11.4.7

♦ (CHANGE) Add a new clause 11.4.7 after clause 11.4.6 as follows:

11.4.7 Subtype constraint interfaces

When a subtype constraint is interfaced nothing is implicitly interfaced.

The constraints specified in the interfaced subtype constraint are reformed upon interfacing so as not to allow any complex entity data type to be instantiated in the current schema that was not allowed in the foreign schema (see annex C).

Clause 12.2.1

♦ (CHANGE) Replace the paragraph after the list of value-comparison operators with the following paragraph:

These operators may be applied to numeric, logical, string and binary operands. These operators may also be applied to enumeration items declared in enumerations that are not extensible and that are not based on extensible enumerations. In addition, = and <> may be applied to values of aggregate and entity data types and enumeration items declared in extensible enumerations or enumerations based on extensible enumerations.

Clause 12.2.1.5

♦ (CHANGE) Replace the text in clause 12.2.1.5 with the following text:

Value comparison of enumeration items in an enumeration that is not extensible and that is not based on an extensible enumeration is based on their relative positions in the declaration of the enumeration data type. See rule (d) in 8.4.1.

Value comparison of enumeration items in extensible enumerations and enumerations based on extensible enumerations shall be based on comparing the simple or prefixed reference to the enumeration items (see 12.7.1 and 12.7.2).

Clause 12.11

♦ (CHANGE) In the second list item in the list after the first paragraph change the following:

(including defined data types which use a defined data type as the underlying data type)

to:

(including defined data types which use a defined data type as the underlying data type and constructed data types that are based on extensible data types)

Clause 13.3

♦ (CHANGE) Replace the third list item dealing with assignment compatibility of that starts with the text "the declared data type of the variable being assigned to is a defined data type whose fundamental data type is a select data type" with the following list item:

- the declared data type of the variable being assigned to is a defined data type whose fundamental data type is a select data type, and the expression evaluates to a value of a data type which is assignment compatible with one, or more, of the data types specified in the domain of the select data type (including items added to that domain by other select data types based on the select data type).

Clause 15.25

♦ (CHANGE) Replace the first list item (a) following the paragraph describing "Result" with the following:

a. The result set is initialized using the type name V belongs to (by declaration), including the schema name when the type name is a named data type.
   - if V is a formal parameter, it is replaced by the corresponding actual parameter first;
   - if V is an aggregate value, the type name is just the name of the aggregation data type (ARRAY, BAG, LIST, SET), Stop.
   - if V is an enumeration data type based on another enumeration data type add the names of the enumeration data types reached by traversing the based_on relationships from the current enumeration data type;
   - if V is an extensible enumeration data type recursively add the names of the enumeration data types which are extensions to V;

     NOTE - Both of the above statements are true for an extensible enumeration data type which is based on another enumeration data type.
   - if V evaluates to indeterminate (?) an empty set is returned.

♦ (CHANGE) Replace the list item (b.3) following the paragraph describing "Result" with the following:

(3) Repeat for all names in the result set:

- for each select data type in which the current name appears in the select list do the following:
  - add the name of the select data type to the list;
  - if the select data type is based on another select data type, add the names of the select data types reached by traversing the based_on relationships from the current select data type;
  - if the select data type is an extensible select data type recursively add the names of the select data types which are extensions to the current select data type.

Clause A.1.1

♦ (CHANGE) Add the following after rule 118:

```
350 BASED_ON = 'based_on' .

351 END_SUBTYPE_CONSTRAINT = 'end_subtype_constraint' .

352 EXTENSIBLE = 'extensible' .

353 GENERIC_ENTITY = 'generic_entity' .
```

```
354 RENAMED = 'renamed' .

355 SUBTYPE_CONSTRAINT = 'subtype_constraint' .

356 TOTAL_OVER = 'total_over' .

357 WITH = 'with' .
```

Clause A.2

♦ (CHANGE) Add the following after rule 318:

```
400 abstract_supertype = ABSTRACT SUPERTYPE ';' .

401 concrete_types = aggregation_types | simple_types | named_types .

402 enumeration_items = '(' enumeration_id { ',' enumeration_id } ')' .

403 enumeration_extension = BASED_ON type_ref [ WITH enumeration_items ] .

404 generic_entity_type = GENERIC_ENTITY .

405 language_version_id = '{ iso standard 10303 part (11) version (4) }' .

406 redeclared_attribute = qualified_attribute [RENAMED attribute_id ] .

407 schema_version_id = string_literal .

408 schema_version_ref = string_literal .

409 select_extension = BASED_ON type_ref [ WITH select_list ] .

410 select_list = '(' named_types { ',' named_types } ')' .

411 subtype_constraint_body = [ abstract_supertype ] [ total_over ] [
supertype_expression ';' ] .

412 subtype_constraint_decl = subtype_constraint_head subtype_constraint_body
END_SUBTYPE_CONSTRAINT ';' .

413 subtype_constraint_head = SUBTYPE_CONSTRAINT subtype_constraint_id FOR
entity_ref ';' .

414 subtype_constraint_id = simple_id .

415 total_over = TOTAL_OVER '(' entity_ref { ',' entity_ref } ')' ';' .
```

Clause A.2

♦ (CHANGE) Replace the each of the like numbered rules as follows:

```
167 attribute_decl = attribute_id | redeclared_attribute .

171 base_type = concrete_types | generalized_types .

189 declaration = entity_decl | subtype_constraint_decl | function_decl |
procedure_decl | type_decl .

197 entity_head = ENTITY entity_id [ ABSTRACT ] [ subsuper ] ';' .

201 enumeration_type = [ EXTENSIBLE ] ENUMERATION [ (( OF enumeration_items ) |
enumeration_extension) ] .

211 generalized_types = aggregate_type | general_aggregation_type | generic_type |
generic_entity_type .

267 reference_clause = REFERENCE FROM schema_ref [ schema_version_ref ] [ '('
resource_or_rename { ',' resource_or_rename } ')' ] ';' .

281 schema_decl = SCHEMA schema_id [ schema_version_id ] ';' schema_body END_SCHEMA
';' .

284 select_type = [ EXTENSIBLE ] [ GENERIC_ENTITY ] SELECT [( select_list |
select_extension ) ] .

302 syntax = [ language_version_id ] schema_decl { schema_decl } .

313 use_clause = USE FROM schema_ref [ schema_version_ref ] [ '('
named_type_or_rename { ',' named_type_or_rename } ')' ] ';' .
```

Clause A.3

♦ (CHANGE) Add the following to the cross reference listing after the last production, numbered 318, in the listing:

```
350 BASED_ON                | 403 409
351 END_SUBTYPE_CONSTRAINT  | 412
352 EXTENSIBLE              | 201 284
353 GENERIC_ENTITY          | 284 404
354 RENAMED                 | 406
355 SUBTYPE_CONSTRAINT      | 413
356 TOTAL_OVER              | 415
357 WITH                    | 403 409
400 abstract_supertype      | 411
401 concrete_types          | 171
402 enumeration_items       | 201 403
403 enumeration_extension   | 201
404 generic_entity_type     | 211
405 language_version_id     | 302
406 redeclared_attribute    | 167
407 schema_version_id       | 281
408 schema_version_ref      | 267 313
409 select_extension        | 284
410 select_list             | 284 409
411 subtype_constraint_body | 412
412 subtype_constraint_decl | 189
413 subtype_constraint_head | 412
414 subtype_constraint_id   | 413
415 total_over              | 411
```

♦ (CHANGE) Replace the following like numbered productions in the cross reference listing:

1 ABSTRACT	156 197 400
47 FOR	164 234 278 413
100 SUPERTYPE	156 300 400
140 simple_id	168 187 198 199 210 236 252 260 279 282 305 307 314 414
147 entity_ref	195 219 234 245 254 275 296 301 413 415
154 type_ref	200 245 275 309 403 409
162 aggregation_types	309 401
168 attribute_id	145 167 406
199 enumeration_id	148 201 402
211 generalized_types	171 253
245 named_types	246 253 284 401 410
262 qualified_attribute	266 406
289 simple_types	253 309 401
292 string_literal	238 407 408
298 supertype_expression	250 295 301 411

Clause B.2

♦ (CHANGE) Rename B.2 to the following: Supertype and subtype constraint operators

♦ (CHANGE) Replace the first sentence in the first paragraph with the following:

Using the formalism above, it is possible to rewrite the restrictions defined in EXPRESS supertype expressions and subtype constraints in terms of evaluated sets.

♦ (CHANGE) Replace the first sentence in the second paragraph with the following:

These reductions alone do not completely describe the full meaning of the supertype expression or subtype constraints, in particular the `ONEOF` and `TOTAL_OVER` clauses.

Clause B.3

♦ (CHANGE) Replace clause B.3 with the following:

B.3 Interpreting the possible complex entity data types

The interpretation of the supertype expressions and subtype constraints, with additional information available from the declared structure, allows the developer of an EXPRESS schema to determine the complex entity data types which would be instantiable given those declarations. To enable this determination the evaluated set of complex entity data types for the subtype/supertype graph may be generated. For this purpose the following terms are defined:

Multiply inheriting subtype: A multiply inheriting subtype is one which identifies two or more supertypes in its subtype declaration.

Root supertype: A root supertype is a supertype that is not a subtype.

The evaluated set R of complex entity data types is computed by the following process:

a. Identify all entity declarations which form the subtype/supertype graph.

   NOTE 1 - This may require multiple iterations in cases with complex subtype/supertype graphs.

b. For each supertype i in the subtype/supertype graph within which a supertype constraint is declared, construct a subtype constraint of the form:

```
SUBTYPE_CONSTRAINT i_superconstraint FOR i;
<supertype_constraint> ;
END_SUBTYPE_CONSTRAINT;
```

   replacing `<supertype_constraint>` with the supertype constraint declared in the entity. Consider this constraint as part of the schema for the purposes of this algorithm. Ignore the supertype expression in the entity declaration that was the basis for the subtype constraint for the purposes of this algorithm.

   NOTE 2 - This step converts supertype contraints declared in the entity to equivalent subtype contraint declarations.

c. For each supertype i in the subtype/supertype graph, identify all data types j1, j2, ... jk in the subtype/supertype graph that are defined as subtypes of i but do not occur in any subtype constraint defined for i in the schema or generated in step (b) and construct a subtype constraint of the form:

```
SUBTYPE_CONSTRAINT i_othersubtypes FOR i;
j1 ANDOR j2 ANDOR ... ANDOR jk;
END_SUBTYPE_CONSTRAINT;
```

   Consider this constraint as part of the schema for the purposes of this algorithm.

d. For each supertype i in the subtype/supertype graph identify all subtype constraints sc_1 , sc_2 , ... sc_k , that have i in their `FOR` clause. At this point the parts of subtype constraints that contain total coverage or abstract restrictions are ignored. Combine the subtype expressions sx_i of these constraints into a single subtype constraint st_i of the form: (sx_1 andor sx_2 andor sx_3 ... andor sx_k)

e. For each supertype i in the subtype/supertype graph, generate the evaluated set which represents the constraints between its immediate subtypes by applying the reductions in B.2 and the identities in B.1 to the subtype constraint st_i given by step d above. Combine i with the result using the & operator. If i is not defined as an abstract supertype in its entity declaration or in any subtype constraint of i then add i to the result using +. Call this set E_i .

f. For each root supertype r in the subtype/supertype graph, expand E_r as follows:
   1. For each subtype s of r, replace every occurrence of s (including those within partial complex entity data types) in E_r with E_s , if available, and apply reductions in B.2 and the identities in B.1.
   2. Recursively apply step (f.1) to each s, expanding subtypes of s until leaf entities are reached (for which no E_s is available).

   NOTE 2 - This recursion must terminate, since there are no cycles in the subtype/supertype graph.

g. Combine root sets. Create $R = \sum_r E_r$ that is $E_{r1} + E_{r2} + ...$, i.e. R is the union of the sets produced in step (f).
h. For each supertype s in R, for each total coverage subtype constraint t_1 , t_2 , ... t_k defined for s
   1. Define t to be: (t_1 andor t_2 ... andor t_k)
   2. For all immediate subtypes s_i of s not in {t_1 , t_2 , ... t_k } replace each occurrence of s_i in R by the expression derived from (s_i and t) using the definitions in B.2.2.
   3. Reduce R according to the reductions in B.2 and the identities in B.1.
i. For each multiply inheriting subtype m, do the following:
   1. For each of its immediate supertypes s, generate the set R/m/s which contains exactly those complex data types in R which include both m and s.
   2. Generate the evaluated set of supertype combinations permitted by m, $P_m = R/m/s_1$ & $R/m/s_2$ &..., i.e., combine the evaluated sets produced in step (i.1) using &.
   3. Generate the evaluated set of supertype combinations which may not include all the supertypes of m, $X_m = \sum_s R/m/s$, i.e., union together the evaluated sets produced in step (i.1).
   4. Put $R = (R - X_m) + P_m$.
j. For each subtype constraint expression k (including those generated in steps (b) and (h)) of the form oneof(S_1 , S_2 , ...), do the following:
   1. For each pair of subexpressions S_i , S_j controlled by k (i < j), compute the set of combinations disallowed by oneof (S_i , S_j): $D_k^{i,j} = [S_i \& S_j]$. Reduce $D_k^{i,j}$ according to the reductions in B.2 and the identities in B.1.
   2. Set $D_k = \sum_{i,j} D_k^{i,j}$, i.e. D_k is the union of the sets computed in step (j.1).
   3. Put $R = R - (R / D_k)$.
k. For each subtype constraint expression k (including those generated in steps (b) and (h)) of the form S_1 andS_2 , do the following:
   1. Compute the set of required combinations dictated by k, $Q_k = [S_1 \& S_2]$. Reduce Q_k according to the reductions in B.2 and the identities in B.1.
   2. For each entity data type i named in k, compute the set of invalid entity combinations containing i which are disallowed by k, $D_k^i = R/i - R/(Q_k /i)$.
   3. Set $D_k = \sum_i D_k^i$, i.e., D_k is the union of the sets computed in step (k.2).
   4. Put $R = R - D_k$.
l. The final evaluated set R is the evaluated set for the input subtype/supertype graph.

EXAMPLE 155 - In this example, only the entity supertype and subtype declarations are given as this is all the information required to interpret the possible complex entity data types.

```
SCHEMA example;

ENTITY p;
END_ENTITY;

SUBTYPE_CONSTRAINT p_subs FOR p;
ONEOF(m, f) AND ONEOF(c, a);
END_SUBTYPE_CONSTRAINT;

ENTITY m SUBTYPE OF (p);
END_ENTITY;

ENTITY f SUBTYPE OF (p);
END_ENTITY;

ENTITY c SUBTYPE OF (p);
END_ENTITY;

ENTITY a ABSTRACT SUBTYPE OF (p);
END_ENTITY;

SUBTYPE_CONSTRAINT no_li FOR a;
ONEOF(l, i);
END_SUBTYPE_CONSTRAINT;

ENTITY l SUBTYPE OF (a);
END_ENTITY;

ENTITY i SUBTYPE OF (a);
END_ENTITY;


END_SCHEMA;
```

This schema is represented by EXPRESS-G in figure B.1.

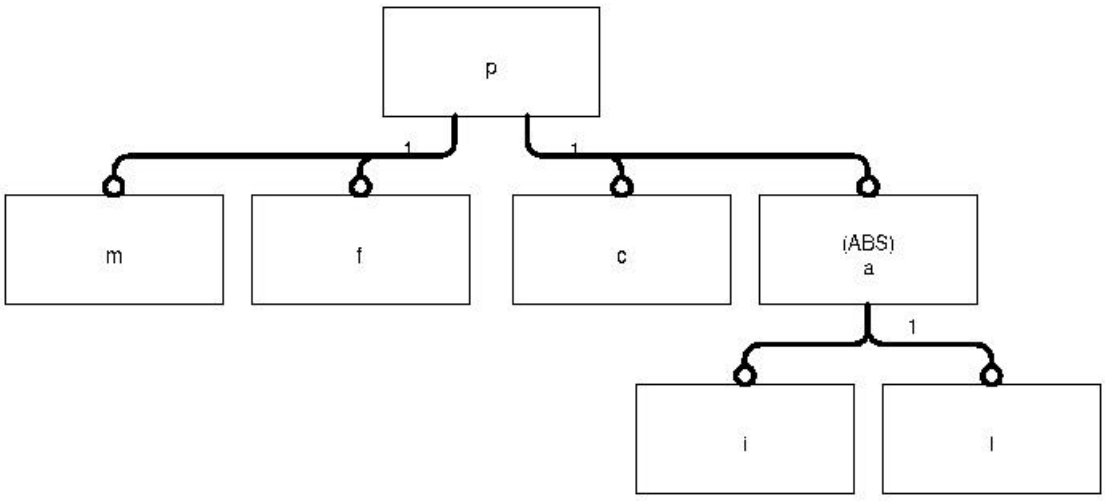

Figure B.1 - EXPRESS-G diagram of schema for example 155.

The potential complex entity data types can be determined as follows:

- The above EXPRESS already gives us all of the entity declarations and complete supertype expressions as required in steps (a), (b) and (c).
- Applying step (d) gives:

```
SUBTYPE_CONSTRAINT stp FOR p;
  ( (ONEOF(m,f)) ANDOR ((ONEOF(a,c)) );
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT sta FOR a;
  (ONEOF(i,l));
END_SUBTYPE_CONSTRAINT;
```

- Applying step (e) gives:
- E_p [p&m&c, p&m&a, p&f&c, p&f&a, p]
- E_a [a&l, a&i]
- Applying step (f) expands the declarations of the root entities, in this case p. The resulting set is:
- E_p = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i,p]
- Combining the root sets in step (g) gives:
- R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p]
- There are no total over constraints or multiply inheriting subtypes, so steps (h) and (i) are not required.
- Applying step (j) to each ONEOF constraint gives:
  - ONEOF(m, f):
  - $D_1^{1,2}$ = [m&f]
  - D_1 = [m&f]
  - Removing D_1 from R according to step (j.3) leaves R unchanged. Thus we are left with:
  - R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i,p]
  - ONEOF (c, a):
  - $D_2^{1,2}$ = [c&a]
  - D_2 = [c&a]
  - Removing D_2 from R according to step (j.3) leaves R unchanged. Thus we are left with:
  - R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i,p]
  - ONEOF (l, i):
  - $D_3^{1,2}$ = [l&i]
  - D_3 = [l&i]
  - Removing D_3 from R according to step (j.3) leaves R unchanged. Thus we are left with:
  - R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i,p]
- Applying step (k) to each AND constraint gives:
  - ONEOF(m, f) AND ONEOF(c, a):
  - Q_1 = [m&c, m&a, f&c, f&a]
  - D_1^m = [ ]
  - D_1^f = [ ]
  - D_1^c = [ ]
  - D_1^a = [ ]
  - D_1 = [ ]
  - Removing D_1 from R according to step (k.4) leaves R unchanged. Thus we are left with:
  - R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i,p]
- According to step (l), the result is thus:
- R = [p&m&c, p&m&a&l, p&m&a&i, p&f&c, p&f&a&l, p&f&a&i, p]

This example, although apparently arbitrary, could have been made more realistic if the entities had been given more

descriptive names. For instance, if instead of `p`, `m`, `f`, `c`, `a`, `l` and `i` the entities were respectively called, `person`, `male`, `female`, `citizen`, `alien`, `legal_alien` and `illegal_alien`.

With this interpretation, reading off a few of the items in the final evaluated set gives:

- A `person` who is `male` and a `citizen`.
- A `person` who is a `male` `alien` and who is a `legal_alien`.
- A `person` who is a `male` `alien` and who is an `illegal_alien`.
- A `person` who ...
- A `person`.

EXAMPLE 156 - This example demonstrates that `ONEOF` is a global constraint which cannot be overridden by multiple inheritance.

```
SCHEMA diamond;

ENTITY a;
END_ENTITY;

SUBTYPE_CONSTRAINT a_subs FOR a;
ONEOF(b, c);
END_SUBTYPE_CONSTRAINT;

ENTITY b SUBTYPE OF (a);
END_ENTITY;

ENTITY c SUBTYPE OF (a);
END_ENTITY;

ENTITY d SUBTYPE OF (b, c);
END_ENTITY;

END_SCHEMA;
```

This schema is represented by EXPRESS-G in figure B.2.

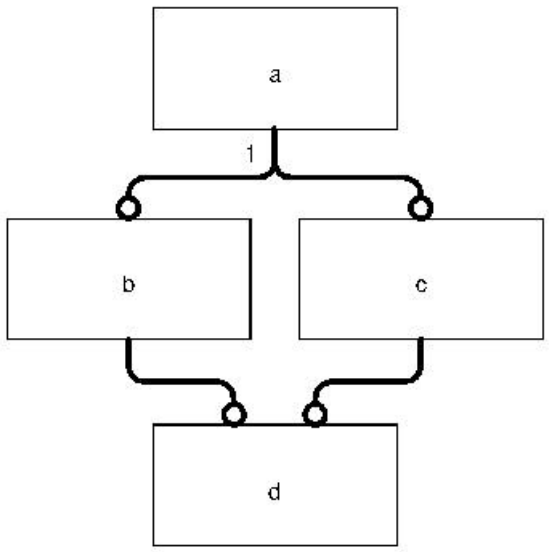

Figure B.2 - EXPRESS-G diagram of schema for example 156.

The potential complex entity data types can be determined as follows:

- The above EXPRESS already gives us all of the entity declarations required in step (g).
- Step (b) creates:
```
SUBTYPE_CONSTRAINT a_superconstraint FOR a;
ONEOF(b,c);
END_SUBTYPE_CONSTRAINT;
```
- Step (c) creates:
```
SUBTYPE_CONSTRAINT b_othersubtypes FOR b;
d;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT c_othersubtypes FOR c;
d;
END_SUBTYPE_CONSTRAINT;
```
- Step (d) creates:
```
SUBTYPE_CONSTRAINT sc_a FOR a;
ONEOF(b,c);
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT sc_b FOR b;
d;
END_SUBTYPE_CONSTRAINT;
SUBTYPE_CONSTRAINT sc_c FOR c;
d;
END_SUBTYPE_CONSTRAINT;
```
- Applying step (e) gives:
- E_a [a&b, a&c, a]
- E_b [b&d, b]
- E_c [c&d, c]
- E_d [d]
- Applying step (f) expands the declarations of the root entities, a. The resulting sets are:
- E_a = [a&b&d, a&b, a&c&d, a&c, a]
- Combining the root sets in step (g) gives:
- R = [a&b&d, a&b, a&c&d, a&c, a]
- Applying step (i) to each multiply inheriting subtype gives the following results:
  - For entity d:
  - C_d^b = [a&b&d]
  - C_d^c = [a&c&d]
  - P_d = [a&b&c&d]
  - X_d = [a&b&d, a&c&d]
  - The new set R = (R - X_d) + P_d is then:
  - [a&b, a&c, a, a&b&d&c]
- Applying step (j) to each ONEOF constraint gives:
  - ONEOF(b, c):
  - $D_1^{1,2}$ = [b&c]
  - D_1 = [b&c]
  - Removing D_1 from R according to step (j.3) removes the following elements from R: [a&b&d&c]. Thus we are left with:
  - R = [a&b, a&c, a]
- There are no supertype expressions which use AND, so step (k) is not required.
- According to step (l), the result is thus:
- R = [a&b, a&c, a]

EXAMPLE 157 - This example shows the effect of applying constraints to a complex structure which contains at least

one of each possible type of constraint. This example is not expected to model any useful concept and is used purely to demonstrate the algorithm.

```
SCHEMA complex;

ENTITY a;
END_ENTITY;

ENTITY b SUBTYPE OF (a);
END_ENTITY;

ENTITY c SUBTYPE OF (a);
END_ENTITY;

ENTITY d SUBTYPE OF (a);
END_ENTITY;

ENTITY f SUBTYPE OF (a, z);
END_ENTITY;

ENTITY k SUBTYPE OF (d);
END_ENTITY;

ENTITY l SUBTYPE OF (d, y);
END_ENTITY;

ENTITY x SUBTYPE OF (z);
END_ENTITY;

ENTITY y SUBTYPE OF (z);
END_ENTITY;

ENTITY z;
END_ENTITY;

SUBTYPE_CONSTRAINT a_subs FOR a;
ONEOF(b, c) AND d ANDOR f;
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT d_subs FOR d;
ABSTRACT;
ONEOF(k, l);
END_SUBTYPE_CONSTRAINT;

END_SCHEMA;
```

This schema is represented by EXPRESS-G in B.3.

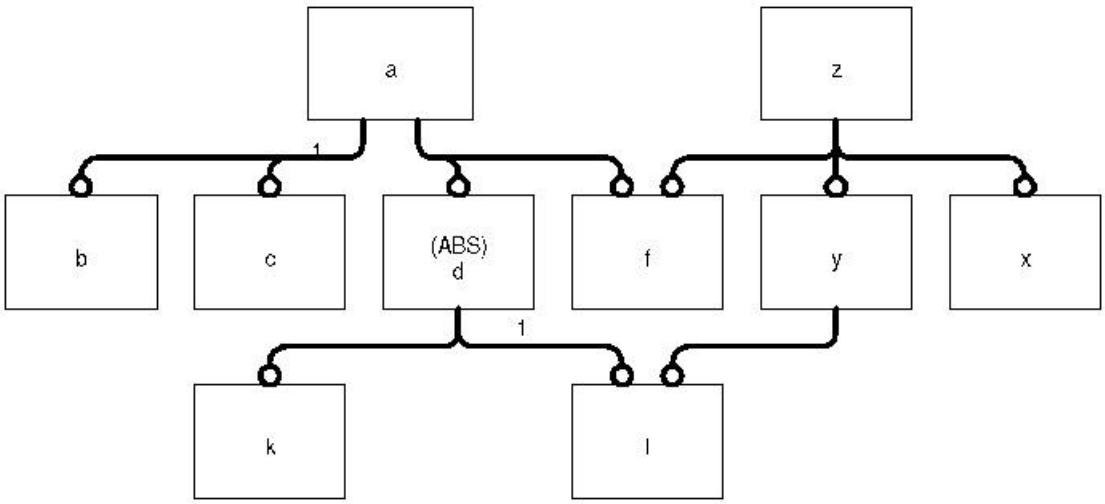

Figure B.3 - EXPRESS-G diagram of schema for example 157.

The potential complex entity data types can be determined as follows:

- The above EXPRESS explicitly specifies all of the entity declarations and complete supertype expressions as required in steps (a) and (b).
- Step (c) creates:

```
SUBTYPE_CONSTRAINT z_othersubtypes FOR z;
f ANDOR y ANDOR x;
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT y_othersubtypes FOR y;
l;
END_SUBTYPE_CONSTRAINT;
```

- Step (d) creates:

```
SUBTYPE_CONSTRAINT st_z FOR z;
f ANDOR y ANDOR x;
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT st_y FOR y;
l;
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT st_a FOR a;
ONEOF(b, c) AND d ANDOR f;
END_SUBTYPE_CONSTRAINT;

SUBTYPE_CONSTRAINT st_d FOR d;
ABSTRACT;>BR> ONEOF(k, l);
END_SUBTYPE_CONSTRAINT;
```

- Applying step (e) gives:
- E_a [a, a&b&d, a&b&d&f, a&c&d, a&c&d&f, a&f]
- E_d [d&k, d&l]
- E_y [l&y, y]
- E_z [f&x&y&z, f&x&z, f&y&z, f&z, x&y&z, x&z, y&z, z]
- Applying step (f) expands the declarations of the root entities, a, z. The resulting sets are:
- E_a = [a, a&b&d&k, a&b&d&l, a&b&d&f&k, a&b&d&f&l,a&c&d&k, a&c&d&l, a&c&d&f&k, a&c&d&f&l, a&f]
- E_z = [f&l&x&y&z,f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z]
- Combining the root sets in step (g) gives:
- R = [a, a&b&d&k, a&b&d&l, a&b&d&f&k, a&b&d&f&l, a&c&d&k, a&c&d&l, a&c&d&f&k, a&c&d&f&l, a&f, f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z]
- There are no total coverage subtype constraints so step (h) is not required.
- Applying step (i) to each multiply inheriting subtype gives the following results:
  - For entity f:
  - C_f^a = [a&b&d&k&f, a&b&d&l&f, a&c&d&k&f, a&c&d&l&f, a&f]
  - C_f^z = [f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z]
  - P_f = [a&b&d&f&k&z, a&b&d&f&k&x&z, a&b&d&f&k&l&y&z, a&b&d&f&k&y&z, a&b&d&f&l&x&y&z, a&b&d&f&k&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&c&d&f&k&z, a&c&d&f&k&x&z, a&c&d&f&k&l&y&z, a&c&d&f&k&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&f&z, a&f&x&z, a&f&l&y&z, a&f&y&z, a&f&l&x&y&z, a&f&x&y&z]
  - X_f = [a&b&d&f&k, a&b&d&f&l, a&c&d&f&k, a&c&d&f&l, a&f, f&l&x&y&z, f&l&y&z, f&x&y&z, f&x&z, f&y&z, f&z]
  - The new set R = (R - X_f) + P_f is then: [a, a&b&d&f&k&z, a&b&d&f&k&x&z, a&b&d&f&k&l&y&z, a&b&d&f&k&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l, a&c&d&f&k&z, a&c&d&f&k&x&z, a&c&d&f&k&l&y&z, a&c&d&f&k&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&k, a&c&d&l, a&f&z, a&f&x&z,

a&f&l&y&z, a&f&y&z, a&f&l&x&y&z, a&f&x&y&z, l&x&y&z, l&y&z, x&y&z, x&z, y&z, z]

- For entity l:
- C_l^d = [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l]
- C_l^y = [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&f&l&y&z, a&f&l&x&y&z, l&x&y&z l&y&z]
- P_l = [a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&d&f&l&y&z, a&b&c&d&f&l&x&y&z, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l&x&y&z, a&c&d&l&y&z]
- X_l = [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&l&z, a&b&d&f&l&x&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&l, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&l&z, a&c&d&f&l&x&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&l, a&f&l&y&z, a&f&l&x&y&z, l&x&y&z, l&y&z]
- The new set R = (R - X_l) + P_l is then: [a, a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&d&f&l&y&z, a&b&c&d&f&l&x&y&z, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z]

- Applying step (j) to each ONEOF constraint gives:
- ONEOF (b, c):
- $D_1^{1,2}$ = [b&c]
- D_1 = [b&c]
- Removing D_1 from R according to step (j.3) removes the following elements from R: [a&b&c&d&f&k&l&y&z, a&b&c&d&f&k&l&x&y&z, a&b&c&d&f&l&y&z, a&b&c&d&f&l&x&y&z]
- Thus we are left with:
- R = [a, a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z]
- ONEOF(k, l):
- $D_2^{1,2}$ = [k&l]
- D_2 = [k&l]
- Removing D_2 from R according to step (j.3) removes the following elements from R: [a&b&d&f&k&l&y&z, a&b&d&f&k&l&x&y&z, a&c&d&f&k&l&y&z, a&c&d&f&k&l&x&y&z]
- Thus we are left with:
- R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z, a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z, a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z]

- Applying step (k) to each AND constraint gives:
- ONEOF (b, c) AND d:
- Q_1 = [b&d, c&d]
- D_1^b = [ ]
- D_1^c = [ ]
- D_1^d = [ ]
- D_1 = [ ]
- Removing D_1 from R according to step (k.4) leaves R unchanged. Thus we are left with:
- R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z,

```
a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z,
a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z,
a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z]
```

- According to step (l), the result is thus:
- ```
  R = [a, a&b&d&f&k&x&z, a&b&d&f&k&y&z, a&b&d&f&k&x&y&z, a&b&d&f&k&z, a&b&d&f&l&y&z,
  a&b&d&f&l&x&y&z, a&b&d&k, a&b&d&l&x&y&z, a&b&d&l&y&z, a&c&d&f&l&y&z, a&c&d&f&l&x&y&z,
  a&c&d&f&k&x&z, a&c&d&f&k&y&z, a&c&d&f&k&x&y&z, a&c&d&f&k&z, a&c&d&k, a&c&d&l&x&y&z,
  a&c&d&l&y&z, a&f&z, a&f&x&z, a&f&y&z, a&f&x&y&z, x&y&z, x&z, y&z, z]
  ```

Annex C

♦ (CHANGE) In the list following the second paragraph add the following as the last list items:

- TOTAL_OVER(A , <>, ...) => TOTAL_OVER(A, ...)

- TOTAL_OVER(<>) => <>

♦ (CHANGE) In the list following the fourth paragraph, remove the list items labelled (c) and (d) and replace them with the following single to list item:

> c) Compute the evaluated set according to the algorighm in annex B starting with step (b) and applying the reductions specified in the first two paragraphs of this annex to the constraints generated in steps (b), (c) and (h) specified in B.3.

Clause D.2.1.1

♦ (CHANGE) Add the following subclause after figure D.3 in D.2.1:

D.2.1.1 Symbols for generalized data types

The symbol for the EXPRESS `GENERIC` data type and `GENERIC_ENTITY` data type is the same as for EXPRESS simple data types. The name of the data type is enclosed within the box as shown in figure D.3a. The symbol for the EXPRESS `AGGREGATE` data type is an open square bracket followed by a closed square bracket.

Figure D.3a - Symbols for generic and generic entity data types.

Clause D.2.2

♦ (CHANGE) After the sentence following figure D.4 add the following sentence as part of the same paragraph:

If the select data type is a generic-entity select data type then the data type name is preceeded by a
  ```

superscripted asterisk (*) symbol.

Clause D.2.2.1

♦ (CHANGE) Add the following new clause D.2.2.1 following the last paragraph in clause D.2.2:

D.2.2.1 Extensible constructed data types

Extensible constructed data types are denoted in EXPRESS-G by placing the characters EX, in parenthesis, i.e. (EX) before the name of the constructed data type as shown in figure D.6a.

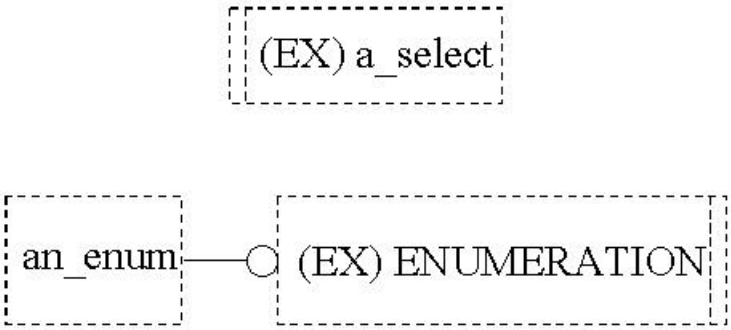

Figure D.6a - Symbols for extensible constructed data types

Clause D.3

♦ (CHANGE) In the paragraph after figure D.10, add the following sentence before the last sentence:

The extension of one constructed data type by another is presented as a thick line.

♦ (CHANGE) In the second paragraph, add the following sentence after the last sentence as part of the same paragraph:

For the extension of constructed data types, the emphasized direction is toward the constructed data type that is based on the extensible data type (i.e., the circle is at the end of the line attached to the constructed data type based on the extensible constructed data type).

Clause D.5.4

♦ (CHANGE) Add the following paragraph, note and figure following note 2:

The extension relationship between an extensible constructed data type and a constructed data type based on it is denoted by a thick line linking the extensible data type to its extensions. Since there may be more than one extension to an extensible data type the link linking the extensible data type to its extensions may branch. The extensible data type may have multiple lines leading to its extensions. The end of the line connected to the extensible data type has no end style. The end of the line connected to the extension is signified by an open circle.

NOTE 3 - Two extensions to an extensible select data type, one of which is itself extensible, can be seen in figure D.14a.

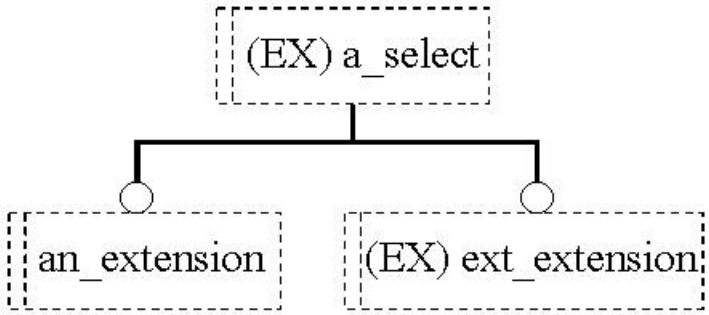

Figure D.14a - Extensible select data type diagram.

Clause D.5.5

♦ (CHANGE) After the last sentence in the first paragraph following the heading "Subtype/supertype" add the following sentence and diagram:

Should the abstract supertype constraint be declared in a subtype constraint this may be represented by a circle surrounding the letters "ABS", as shown in figure D.14b, with an unclosed arrow pointing from the circle to the object to which the constraint is applied.

Figure D.14b - Symbol for abstract supertype in subtype constraint.

♦ (CHANGE) After the first paragraph following the heading "Subtype/supertype" add the following paragraph:

When an entity is declared to be abstract (not an abstract supertype, but an abstract entity), the characters AE, enclosed by parentheses, i.e. (AE), precede the name of the entity within the entity box symbol as shown in figure D.14c.

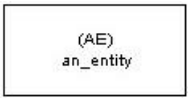

Figure D.14c - Symbol for abstract entity.

♦ (CHANGE) After the last sentence in the second paragraph following the heading "Subtype/supertype" add the following sentence:

The TOTAL_OVER constraint is presented as a circle containing the letter "T", as shown in figure D.14d, with unclosed arrows from the circle to the entity boxes included in the contstraint.

Figure D.14d - Symbol for total over coverage constraint.

♦ (CHANGE) In the paragraph following figure D.15 add the following sentence after the first sentence:

If the attribute redeclaration also includes a rename of the attribute the new name follows the original name, these being separated by the greater than symbol (`>`).

Clause D.7.1

♦ (CHANGE) Add the following list items after item o):

p) All abstract entities are marked;

q) All `TOTAL_OVER` constraints are marked;

r) All relationships between extensible constructed data types and their extensions are displayed;

s) All the new and old names of attributes renamed during redeclaration are displayed;

t) All `GENERIC_ENTITY SELECT` constraints are displayed;

u) All `GENERIC` and `GENERIC_ENTITY` data types used within the schema are displayed.

Annex F

♦ (CHANGE) Replace the text of annex F with the following text:

F.1 Document identification

In order to provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part (11) version (4) }

is assigned to this part of ISO 10303. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

F.2 Syntax identification

In order to provide for unambiguous identification of an information object in an open system, the object identifier

{ iso standard 10303 part (11) version (4) object (1) EXPRESS-syntax (1) }

is assigned to the syntax of EXPRESS. The meaning of this value is defined in ISO/IEC 8824-1, and is described in ISO 10303-1.

Annex K

♦ (CHANGE) Add the following annex after annex J:

Annex K
(informative)

Deprecated features of EXPRESS

Under this amendment to EXPRESS there are several concepts that now have two different syntaxes with which they may be specified. This duplication exists so that existing schemas remain valid while allowing for the use of new constructs. In future editions of EXPRESS, this duplication will be eliminated by the removal of the EXPRESS Edition 1 syntax for constructs for which new syntax has been made available. For this reason, the use of the following syntax is deprecated:

- the abstract supertype denoted by (ABS) in the entity in EXPRESS-G diagrams;
- the `SUPERTYPE` constraint specification syntax which may now be specified by the new `SUBTYPE_CONSTRAINT` declaration.

There are also semantics associated with concepts that exist in EXPRESS Edition 1 that will be modified or removed in a future edition of EXPRESS. For this reason, the following semantics are deprecated:

- the ordering of the enumeration items associated with a non-extensible, non-extension `ENUMERATION` data type;
- the omission of the language version identifier.

Annex L

♦ (CHANGE) Add the following annex after the new annex K:

Annex L
(informative)

Examples of the new EXPRESS constructs

This annex provides examples of the new constructs added to the language as the result of Amendment 1. No claim is made concerning the quality of these models. They are simply examples showing how the new constructs may be used.

L.1 Product management example

Example 177 shows the use of the following constructs in a schema that is a resource to be used by schemas that are more complete with respect to the application domain:

- extensible constructed data types;
- generic-entity constraint on select;
- abstract entity;
- renaming attributes;
- subtype constraint;
- language version.

EXAMPLE 177 - This example uses the extensible constructed data types.

```
{iso standard 10303 part (11) version (4)}
SCHEMA my_product_management;


USE FROM generic_product_management;

TYPE my_additional_categories = ENUMERATION BASED_ON product_category_names WITH ( document,
drawing, electromechanical, mechanical, electrical, pump );
END_TYPE;

TYPE my_additional_values = ENUMERATION BASED_ON approval_status_values WITH ( approved,
disapproved, pending );
END_TYPE;

TYPE my_approvable_objects = EXTENSIBLE SELECT BASED_ON approvable_objects WITH ( product,
product_category, product_to_category_relationship );
END_TYPE;

ENTITY approval_by_person_in_organization
  SUBTYPE OF ( approval );
  SELF\approval.approved_by : person_in_organization;
END_ENTITY;

ENTITY approval_by_person
  SUBTYPE OF ( approval );
  SELF\approval.approved_by : person;
END_ENTITY;

SUBTYPE_CONSTRAINT not_both FOR approval;
  ONEOF ( approval_by_person, approval_by_person_in_organization );
END_SUBTYPE_CONSTRAINT;


END_SCHEMA;


SCHEMA generic_product_management;

TYPE product_category_names = EXTENSIBLE ENUMERATION OF ( part, tool, raw_material );
END_TYPE;

TYPE approval_status_values = EXTENSIBLE ENUMERATION;
END_TYPE;

TYPE approvable_objects = EXTENSIBLE GENERIC_ENTITY SELECT;
END_TYPE;

ENTITY product;
  name : STRING;
```

```
END_ENTITY;


ENTITY product_category;
  name : product_category_names;
END_ENTITY;


ENTITY binary_entity_relationship ABSTRACT;
  end_one : GENERIC_ENTITY;
  end_two : GENERIC_ENTITY;
END_ENTITY;


ENTITY product_to_category_relationship
  SUBTYPE OF ( binary_entity_relationship );
  SELF\binary_entity_relationship.end_one RENAMED the_category : product_category;
  SELF\binary_entity_relationship.end_two RENAMED the_product : product;
END_ENTITY;


ENTITY approval ABSTRACT;
  approved_by : GENERIC;
  status : approval_status_values;
  approved_items : SET[1:?] OF approvable_objects;
END_ENTITY;


ENTITY person;
  name : STRING;
END_ENTITY;


ENTITY organization;
  name : STRING;
END_ENTITY;


ENTITY person_in_organization_relationship
  SUBTYPE OF ( binary_entity_relationship );
  role_of_person : STRING;
  SELF\binary_entity_relationship.end_one RENAMED the_person : person;
  SELF\binary_entity_relationship.end_two RENAMED the_organization : organization;
END_ENTITY;


END_SCHEMA;
```